Discovering Reference Models by Mining Process Variants Using a Heuristic Approach

Chen Li[1] *, Manfred Reichert[2], and Andreas Wombacher[1]

[1] Computer Science Department, University of Twente, The Netherlands
lic@cs.utwente.nl; a.wombacher@utwente.nl
[2] Institute of Databases and Information Systems, Ulm University, Germany
manfred.reichert@uni-ulm.de

Abstract. Recently, a new generation of adaptive Process-Aware Information Systems (PAISs) has emerged, which enables structural process changes during runtime. Such flexibility, in turn, leads to a large number of process variants derived from the same model, but differing in structure. Generally, such variants are expensive to configure and maintain. This paper provides a heuristic search algorithm which fosters learning from past process changes by mining process variants. The algorithm discovers a reference model based on which the need for future process configuration and adaptation can be reduced. It additionally provides the flexibility to control the process evolution procedure, i.e., we can control to what degree the discovered reference model differs from the original one. As benefit, we cannot only control the effort for updating the reference model, but also gain the flexibility to perform only the most important adaptations of the current reference model. Our mining algorithm is implemented and evaluated by a simulation using more than 7000 process models. Simulation results indicate strong performance and scalability of our algorithm even when facing large-sized process models.

1 Introduction

In today's dynamic business world, success of an enterprise increasingly depends on its ability to react to changes in its environment in a quick, flexible and cost-effective way. Generally, process adaptations are not only needed for configuration purpose at build time, but also become necessary for single process instances during runtime to deal with exceptional situations and changing needs [11, 17]. In response to these needs adaptive process management technology has emerged [17]. It allows to configure and adapt process models at different levels. This, in turn, results in large collections of *process variants* created from the same process model, but slightly differing from each other in their structure. So far, only few approaches exist, which utilize the information about these variants and the corresponding process adaptations [4].

* This work was done in the MinAdept project, which has been supported by the Netherlands Organization for Scientific Research under contract number 612.066.512.

Fig. 1 describes the goal of this paper. We aim at learning from past process changes by "merging" process variants into one generic process model, which covers these variants best. By adopting this generic model as new *reference process model* within the Process-aware Information System (PAIS), need for future process adaptations and thus cost for change will decrease. Based on the two assumptions that (1) process models are well-formed (i.e., block-structured like in WS-BPEL) and (2) all activities in a process model have unique labels, this paper deals with the following fundamental research question: *Given a reference model and a collection of process variants configured from it, how to derive a new reference process model by performing a sequence of change operations on the original one, such that the average distance between the new reference model and the process variants becomes minimal?*

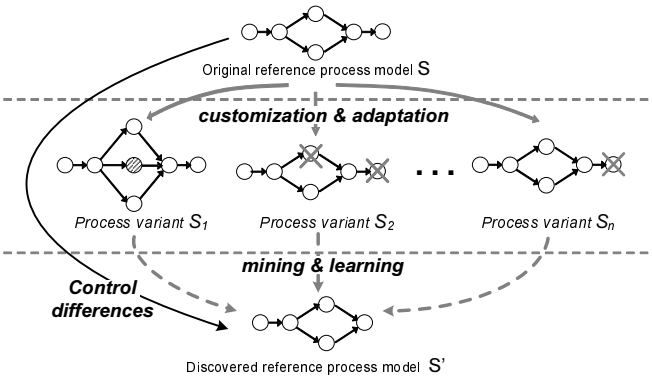

Fig. 1. Discovering a new reference model by learning from past process configurations

The distance between the reference process model and a process variant is measured by the number of high-level change operations (e.g., to insert, delete or move activities [11]) needed to transform the reference model into the variant. Clearly, the shorter the distance is, the less the efforts needed for process adaptation are. Basically, we obtain a new reference model by performing a sequence of change operations on the original one. In this context, we provide users the flexibility to control the distance between old reference model and newly discovered one, i.e., to choose how many change operations shall be applied. Clearly, the most relevant changes (which significantly reduce the average distance) should be considered first and the less important ones last. If users decide to ignore less relevant changes in order to reduce the efforts for updating the reference model, overall performance of our algorithm with respect to the described research goal is not influenced too much. Such flexibility to control the difference between the original and the discovered model is a significant improvement when compared to our previous work [5, 9].

Section 2 gives background information for understanding this paper. Section 3 introduces our heuristic algorithm and provides an overview on how it can be used for mining process variants. We describe two important aspects of our

heuristics algorithm (i.e., fitness function and search tree) in Sections 4 and 5. To evaluate its performance, we conduct a simulation in Section 6. Section 7 discusses related work and Section 8 concludes with a summary.

2 Backgrounds

Process Model. Let $\mathcal{P}$ denote the set of all sound process models. A particular *process model* $S = (N, E, \ldots) \in \mathcal{P}$ is defined as Well-structured Activity Net[3][11]. N constitutes the set of process activities and E the set of control edges (i.e., precedence relations) linking them. To limit the scope, we assume Activity Nets to be block-structured (like BPEL). Examples are depicted in Fig. 2.

Process change. A *process change* is accomplished by applying a sequence of *change operations* to the process model S over time [11]. Such change operations modify the initial process model by altering the set of activities and their order relations. Thus, each application of a change operation results in a new process model. We define *process change* and *process variants* as follows:

Definition 1 (Process Change and Process Variant). *Let $\mathcal{P}$ denote the set of possible process models and $\mathcal{C}$ be the set of possible process changes. Let $S, S' \in \mathcal{P}$ be two process models, let $\Delta \in \mathcal{C}$ be a process change expressed in terms of a high-level change operation, and let $\sigma = \langle \Delta_1, \Delta_2, \ldots \Delta_n \rangle \in \mathcal{C}^*$ be a sequence of process changes performed on initial model S. Then:*

- *$S[\Delta\rangle S'$ iff Δ is applicable to S and S' is the (sound) process model resulting from application of Δ to S.*
- *$S[\sigma\rangle S'$ iff $\exists\ S_1, S_2, \ldots S_{n+1} \in \mathcal{P}$ with $S = S_1$, $S' = S_{n+1}$, and $S_i[\Delta_i\rangle S_{i+1}$ for $i \in \{1, \ldots n\}$. We denote S' as variant of S.*

Examples of high-level change operations include *insert activity*, *delete activity*, and *move activity* as implemented in the ADEPT change framework [11]. While *insert* and *delete* modify the set of activities in the process model, *move* changes activity positions and thus the order relations in a process model. For example, operation $move(S, \texttt{A}, \texttt{B}, \texttt{C})$ shifts activity `A` from its current position within process model S to the position after activity `B` and before activity `C`. Operation $delete(S, \texttt{A})$, in turn, deletes activity `A` from process model S. Issues concerning the correct use of these operations, their generalizations, and formal pre-/post-conditions are described in [11]. Though the depicted change operations are discussed in relation to our ADEPT approach, they are generic in the sense that they can be easily applied in connection with other process meta models as well [17]; e.g., life-cycle inheritance known from Petri Nets [15]. We refer to ADEPT since it covers by far most high-level change patterns and change support features [17], and offers a fully implemented process engine.

[3] A formal definition of a Well-structured Activity Net contains more than only node set N and edge set E. We omit other components since they are not relevant in the given context [18].

Definition 2 (Bias and Distance). *Let S, $S' \in \mathcal{P}$ be two process models.* ***Distance*** *$d_{(S,S')}$ between S and S' corresponds to the minimal number of high-level change operations needed to transform S into S'; i.e., we define $d_{(S,S')} := min\{|\sigma| \mid \sigma \in \mathcal{C}^* \wedge S[\sigma\rangle S'\}$. Furthermore, a sequence of change operations σ with $S[\sigma\rangle S'$ and $|\sigma| = d_{(S,S')}$ is denoted as* ***bias*** *$B_{(S,S')}$ between S and S'.*

The *distance* between S and S' is the minimal number of high-level change operations needed for transforming S into S'. The corresponding sequence of change operations is denoted as *bias* $B_{(S,S')}$ between S and S'.[4] Usually, such distance measures the complexity for model transformation (i.e., configuration). As example take Fig. 2. Here, distance between model S and variant S_1 is 4, i.e., we minimally need to perform 4 changes to transform S into S' [7]. In general, determining bias and distance between two process models has complexity at $\mathcal{NP} - hard$ level [7]. We consider high-level change operations instead of change primitives (i.e., elementary changes like adding or removing nodes / edges) to measure distance between process models. This allows us to guarantee soundness of process models and provides a more meaningful measure for distance [7, 17].

Trace. A *trace* t on process model $S = (N, E, \ldots) \in \mathcal{P}$ denotes a valid and complete execution sequence $t \equiv < a_1, a_2, \ldots, a_k >$ of activity $a_i \in N$ according to the control flow set out by S. All traces S can produce are summarized in trace set $\mathcal{T}_S$. $t(a \prec b)$ is denoted as precedence relation between activities a and b in trace $t \equiv < a_1, a_2, \ldots, a_k >$ iff $\exists i < j : a_i = a \wedge a_j = b$.

Order Matrix. One key feature of any change framework is to maintain the structure of the unchanged parts of a process model [11]. To incorporate this in our approach, rather than only looking at direct predecessor-successor relation between activities (i.e., control edges), we consider the transitive control dependencies for each activity pair; i.e., for given process model $S = (N, E, \ldots) \in \mathcal{P}$, we examine for every pair of activities $a_i, a_j \in N$, $a_i \neq a_j$ their transitive order relation. Logically, we determine order relations by considering all traces the process model can produce. Results are aggregated in an order matrix $A_{|N|\times|N|}$, which considers four types of control relations (cf. Def. 3):

Definition 3 (Order matrix). *Let $S = (N, E, \ldots) \in \mathcal{P}$ be a process model with $N = \{a_1, a_2, \ldots, a_n\}$. Let further $\mathcal{T}_S$ denote the set of all traces producible on S. Then: Matrix $A_{|N|\times|N|}$ is called* ***order matrix*** *of S with A_{ij} representing the order relation between activities a_i,$a_j \in N$, $i \neq j$ iff:*

- *A_{ij} = '1' iff [$\forall t \in \mathcal{T}_S$ with $a_i, a_j \in t \Rightarrow t(a_i \prec a_j)$]. If for all traces containing activities a_i and a_j, a_i always appears* BEFORE *a_j, we denote A_{ij} as '**1**', i.e., a_i always precedes a_j in the flow of control.*
- *A_{ij} = '0' iff [$\forall t \in \mathcal{T}_S$ with $a_i, a_j \in t \Rightarrow t(a_j \prec a_i)$]. If for all traces containing activities a_i and a_j, a_i always appears* AFTER *a_j, we denote A_{ij} as a '**0**', i.e. a_i always succeeds a_j in the flow of control.*

[4] Generally, it is possible to have more than one minimal set of change operations to transform S into S', i.e., given process models S and S' their bias does not need to be unique. A detailed discussion of this issue can be found in [15, 7].

- A_{ij} = '*' *iff* $[\exists t_1 \in \mathcal{T}_S$, *with* $a_i, a_j \in t_1 \wedge t_1(a_i \prec a_j)] \wedge [\exists t_2 \in \mathcal{T}_S$, *with* $a_i, a_j \in t_2 \wedge t_2(a_j \prec a_i)]$. *If there exists at least one trace in which* a_i *appears before* a_j *and another trace in which* a_i *appears after* a_j, *we denote* A_{ij} *as '**', i.e.* a_i *and* a_j *are contained in different parallel branches.*
- A_{ij} = '-' *iff* $[\neg\exists t \in \mathcal{T}_S : a_i \in t \wedge a_j \in t]$. *If there is no trace containing both activity* a_i *and* a_j, *we denote* A_{ij} *as '-', i.e.* a_i *and* a_j *are contained in different branches of a conditional branching.*

Given a process model $S = (N, E, \ldots) \in \mathcal{P}$, the complexity to compute its order matrix $A_{|N|\times|N|}$ is $\mathcal{O}(2|N|^2)$ [7]. Regarding our example from Fig. 2, the order matrix of each process variant S_i is presented on the top of Fig. 4.[5] Variants S_i contain four kinds of control connectors: AND-Split, AND-Join, XOR-Split, and XOR-join. The depicted order matrices represent all possible order relations. As example consider S_4. Activities `H` and `I` never appear in same trace since they are contained in different branches of an XOR block. Therefore, we assign '-' to matrix element A_{HI} for S_4. If certain conditions are met, the order matrix can uniquely represent the process model. Analyzing its order matrix (cf. Def. 3) is then sufficient in order to analyze the process model [7].

It is also possible to handle loop structures based on an extension of order matrices, i.e., we need to introduce two additional order relations to cope with loop structures in process models [7]. However, since activities within a loop structure can run an arbitrarily number of times, this complicates the definition of order matrix in comparison to Def. 3. In this paper, we use process models without loop structures to illustrate our algorithm. It will become clear in Section 4 that our algorithm can easily be extended to also handle process models with loop structures by extending Def. 3.

3 Overview of our Heuristic Search Algorithm

Running Example. An example is given in Fig. 2. Out of original reference model S, six different process variants $S_i \in \mathcal{P}$ $(i = 1, 2, \ldots 6)$ are configured. These variants do not only differ in structure, but also in their activity sets. For example, activity `X` appears in 5 of the 6 variants (except S_2), while `Z` only appears in S_5. The 6 variants are further weighted based on the number of process instances created from them; e.g., 25% of all instances were executed according to variant S_1, while 20% ran on S_2. We can also compute the distance (cf. Def. 2) between S and each variant S_i. For example, when comparing S with S_1 we obtain distance 4 (cf. Fig. 2); i.e., we need to apply 4 high-level change operations [$move(S, \texttt{H}, \texttt{I}, \texttt{D})$, $move(S, \texttt{I}, \texttt{J}, endFlow)$, $move(S, \texttt{J}, \texttt{B}, endFlow)$ and $insert(S, \texttt{X}, \texttt{E}, \texttt{B})$] to transform S into S_1. Based on weight w_i of each variant S_i, we can compute average weighted distance between reference model S and its variants. As distances between S and S_i we obtain 4 for $i = 1, \ldots, 6$ (cf. Fig. 2). When considering variant weights, as average weighted distance, we obtain $4 \times 0.25 + 4 \times 0.2 + 4 \times 0.15 + 4 \times 0.1 + 4 \times 0.2 + 4 \times 0.1 = 4.0$. This means we need to perform on average 4.0 change operations to configure a process variant (and

[5] Due to lack of space, we only depict order matrices for activities `H,I,J,X,Y` and `Z`.

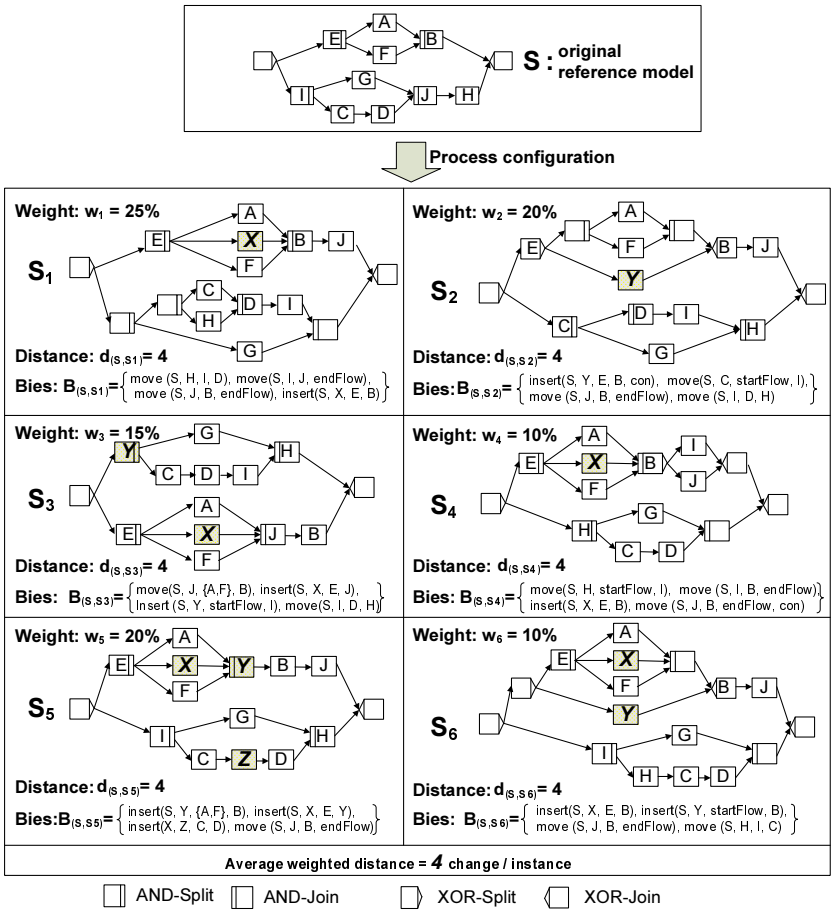

Fig. 2. Illustrating example

related instance respectively) out of S. Generally, *average weighted distance* between a reference model and its process variants represents how "*close*" they are. The goal of our mining algorithm is to discover a reference model for a collection of (weighted) process variants with minimal average weighted distance.

Heuristic Search for Mining Process Variants. As measuring distance between two models has $\mathcal{NP}-hard$ complexity (cf. Def. 2), our research question (i.e., finding a reference model with minimal average weighted distance to the variants), is a $\mathcal{NP}-hard$ problem as well. When encountering real-life cases, finding "the optimum" would be either too time-consuming or not feasible. In this paper, we therefore present a *heuristic search algorithm* for mining variants.

Heuristic algorithms are widely used in various fields, e.g., artificial intelligence [10] and data mining [14]. Although they do not aim at finding the "real optimum" (i.e., it is neither possible to theoretically prove that discovered results are optimal nor can we say how close they are to the optimum), they are

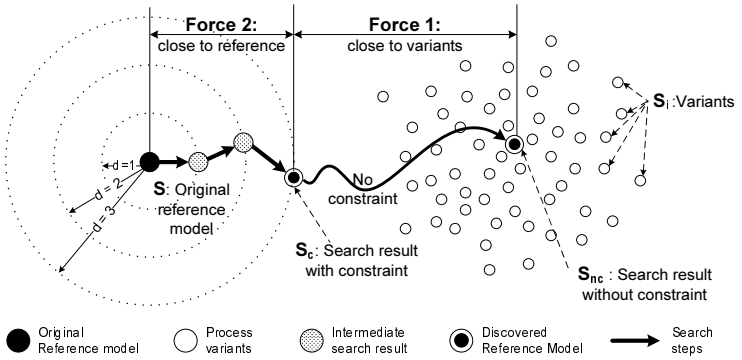

Fig. 3. Our heuristic search approach

widely used in practice. Particularly, they can nicely balance goodness of the discovered solution and time needed for finding it [10].

Fig. 3 illustrates how heuristic algorithms can be applied for the mining of process variants. Here we represent each process variant S_i as single node (white node). The goal for variant mining is then to find the "center" of these nodes (bull's eye S_{nc}), which has minimal average distance to them. In addition, we want to take original reference model S (solid node) into account, such that we can control the difference between the newly discovered reference model and the original one. Basically, this requires us to balance two forces: one is to bring the newly discovered reference model closer to the variants; the other one is to "move" the discovered model not too far away from S. Process designers should obtain the flexibility to discover a model (e.g., S_c in Fig. 3), which is closer to the variants on the one hand, but still within a limited distance to the original model on the other hand.

Our heuristic algorithm works as follows: *First*, we use original reference model S as starting point. As *Step 2*, we search for all neighboring models with distance 1 to the currently considered reference process model. If we are able to find a model S_c with lower average weighted distance to the variants, we replace S by S_c. We repeat Step 2 until we either cannot find a better model or the maximally allowed distance between original and new reference model is reached.

For any heuristic search algorithm, two aspects are important: the *heuristic measure* (cf. Section 4) and the *algorithm* (Section 5) that uses heuristics to search the state space.

4 Fitness Function of our Heuristic Search Algorithm

Any fitness function of a heuristic search algorithm should be quickly computable. Average weighted distance itself cannot be used as fitness function, since complexity for computing it is $\mathcal{NP} - hard$. In the following, we introduce a fitness function computable in polynomial time, to approximately measure "closeness" between a candidate model and the collection of variants.

4.1 Activity Coverage

For a candidate process model $S_c = (N_c, E_c, \ldots) \in \mathcal{P}$, we first measure to what degree its activity set N_c covers the activities that occur in the considered collection of variants. We denote this measure as *activity coverage* $AC(S_c)$ of S_c. Before we can compute it, we first need to determine *activity frequency* $g(a_j)$ with which each activity a_j appears within the collection of variants. Let $S_i \in \mathcal{P}$ $i = 1, \ldots, n$ be a collection of variants with weights w_i and activity sets N_i. For each $a_j \in \bigcup N_i$, we obtain $g(a_j) = \sum_{S_i : a_j \in N_i} w_i$. Table 1 shows the frequency of each activity contained in any of the variants in our running example; e.g., `X` is present in 80% of the variants (i.e., in S_1, S_3, S_4, S_5, and S_6), while `Z` only occurs in 20% of the cases (i.e., in S_5).

Activity	`A`	`B`	`C`	`D`	`E`	`F`	`G`	`H`	`I`	`J`	`X`	`Y`	`Z`
$g(a_j)$	1	1	1	1	1	1	1	1	1	1	0.8	0.65	0.2

Table 1. Activity frequency of each activity within the given variant collection

Let $M = \bigcup_{i=1}^{n} N_i$ be the set of activities which are present in at least one variant. Given activity frequency $g(a_j)$ of each $a_j \in M$, we can compute *activity coverage* $AC(S_c)$ of candidate model S_c as follows:

$$AC(S_c) = \frac{\sum_{a_j \in N_c} g(a_j)}{\sum_{a_j \in M} g(a_j)} \tag{1}$$

The value range of $AC(S_c)$ is $[0, 1]$. Let us take original reference model S as candidate model. It contains activities `A, B, C, D, E, F, G, H, I`, and `J`, but does not contain `X, Y` and `Z`. Therefore, its activity coverage $AC(S)$ is 0.858.

4.2 Structure Fitting

Though $AC(S_c)$ measures how representative the activity set N_c of a candidate model S_c is with respect to a given variant collection, it does not say anything about the structure of the candidate model. We therefore introduce *structure fitting* $SF(S_c)$, which measures, to what degree a candidate model S_c structurally fits to the variant collection. We first introduce *aggregated order matrix* and *coexistence matrix* to adequately represent the variants.

Aggregated Order Matrix. For a given collection of process variants, first, we compute the order matrix of each process variant (cf. Def. 3). Regarding our running example from Fig. 2, we need to compute six order matrices (see to of Fig. 4). Note that we only show a partial view on the order matrices here (activities `H, I, J, X, Y` and `Z`) due to space limitations. As the order relation between two activities might be not the same in all order matrices, this analysis does not result in a fixed relation, but provides a distribution for the four types of order relations (cf. Def. 3). Regarding our example, in 65% of all cases `H` succeeds `I` (as in S_2, S_3, S_4 and S_6), in 25% of all cases `H` precedes `I` (as in S_1), and in 10% of all cases `H` and `I` are contained in different branches of an XOR

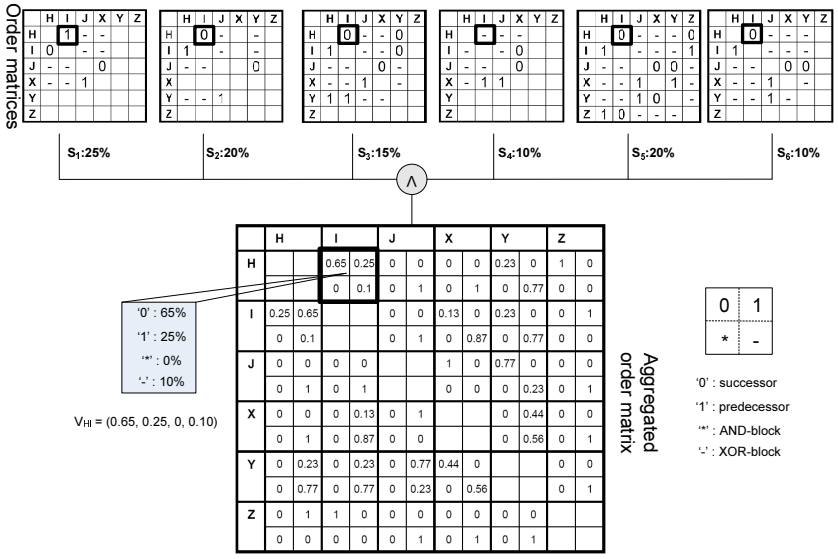

Fig. 4. Aggregated order matrix based on process variants

block (as in S_4) (cf. Fig. 4). Generally, for a collection of process variants we can define the order relation between activities `a` and `b` as 4-dimensional vector $V_{ab} = (v^0_{ab}, v^1_{ab}, v^*_{ab}, v^-_{ab})$. Each field then corresponds to the frequency of the respective relation type ('0', '1', '*' or '-') as specified in Def. 3. For our example from Fig. 2, for instance, we obtain $V_{\texttt{HI}} = (0.65, 0.25, 0, 0.1)$ (cf. Fig. 4). Fig. 4 shows aggregated order matrix V for the process variants from Fig. 2.

Coexistence Matrix. Generally, the order relations computed by an aggregated order matrix may not be equally important; e.g., relationship $V_{\texttt{HI}}$ between `H` and `I` (cf. Fig. 4) would be more important than relation $V_{\texttt{HZ}}$, since `H` and `I` appear together in all six process variants while `H` and `Z` only show up together in variant S_5 (cf. Fig. 2). We therefore define *Coexistence Matrix* CE in order to represent the importance of the different order relations occurring within an aggregated order matrix V. Let S_i $(i = 1 \ldots n)$ be a collection of

	H	I	J	X	Y	Z
H		**1**	**1**	**0.8**	**0.65**	**0.2**
I	**1**		**1**	**0.8**	**0.65**	**0.2**
J	**1**	**1**		**0.8**	**0.65**	**0.2**
X	**0.8**	**0.8**	**0.8**		**0.45**	**0.2**
Y	**0.65**	**0.65**	**0.65**	**0.45**		**0.2**
Z	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	

Fig. 5. Coexistence Matrix

process variants with activity sets N_i and weight w_i. The *Coexistence Matrix* of these process variants is then defined as 2-dimensional matrix $CE_{m\times m}$ with $m = |\bigcup N_i|$. Each matrix element CE_{jk} corresponds to the relative frequency with which activities a_j and a_k appear together within the given collection of variants. Formally: $\forall a_j, a_k \in \bigcup N_i, a_j \neq a_k : CE_{jk} = \sum_{S_i:a_j\in N_i \wedge a_k \in N_i} w_i$. Table 5 shows the coexistence matrix for our running example (partial view).

Structure Fitting $SF(S_c)$ of Candidate Model S_c. Since we can represent candidate process model S_c by its corresponding order matrix A_c (cf. Def. 3), we determine *structure fitting* $SF(S_c)$ between S_c and the variants by measuring how similar order matrix A_c and aggregated order matrix V (representing the variants) are. We can evaluate S_c by measuring the order relations between every pair of activities in A_c and V. When considering reference model S as candidate process model S_c (i.e., $S_c = S$), for example, we can build an aggregated order matrix V^c purely based on S_c, and obtain $V^c_{\texttt{HI}} = (1, 0, 0, 0)$; i.e., `H` always succeeds `I`. Now, we can compare $V_{\texttt{HI}} = (0.65, 0.25, 0, 0.1)$ (representing the variants) and $V^c_{\texttt{HI}}$ (representing the candidate model).

We use Euclidean metrics $f(\alpha, \beta)$ to measure closeness between vectors $\alpha = (x_1, x_2, ..., x_n)$ and $\beta = (y_1, y_2, ..., y_n)$: $f(\alpha, \beta) = \frac{\alpha\cdot\beta}{|\alpha|\times|\beta|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\times\sqrt{\sum_{i=1}^{n} y_i^2}}$. $f(\alpha, \beta) \in [0, 1]$ computes the cosine value of the angle θ between vectors α and β in Euclidean space. The higher $f(\alpha, \beta)$ is, the more α and β match in their directions. Regarding our example we obtain $f(V_{\texttt{HI}}, V^c_{\texttt{HI}}) = 0.848$. Based on $f(\alpha, \beta)$, which measures *similarity* between the order relations in V (representing the variants) and in V^c (representing candidate model), and Coexistence matrix CE, which measures *importance* of the order relations, we can compute *structure fitting $SF(S_c)$ of candidate model S_c* as follows:

$$SF(S_c) = \frac{\sum_{j=1}^{n}\sum_{k=1,k\neq j}^{n}[f(V_{a_j a_k}, V^c_{a_j a_k})\cdot CE_{a_j a_k}]}{n(n-1)} \tag{2}$$

$n = |N_c|$ corresponds to the number of activities in candidate model S_c. For every pair of activities $a_j, a_k \in N_c, j \neq k$, we compute similarity of corresponding order relations (as captured by V and V_c) by means of $f(V_{a_j a_k}, V^c_{a_j a_k})$, and the importance of these order relations by $CE_{a_j a_k}$. Structure fitting $SF(S_c) \in [0, 1]$ of candidate model S_c then equals the average of the similarities multiplied by the importance of every order relation. For our example from Fig. 2, structure fitting $SF(S)$ of original reference model S is 0.749.

4.3 Fitness Function

So far, we have described the two measurements *activity coverage* $AC(S_c)$ and *structure fitting* $SF(S_c)$ to evaluate a candidate model S_c. Based on them, we can compute *fitness* $Fit(S_c)$ of S_c : $Fit(S_c) = AC(S_c) \times SF(S_c)$.

As $AC(S_c) \in [0, 1]$ and $SF(S_c) \in [0, 1]$, value range of $Fit(S_c)$ is [0,1] as well. Fitness value $Fit(S_c)$ indicates how "close" candidate model S_c is to the given collection of variants. The higher $Fit(S_c)$ is, the closer S_c is to the variants and the less configuration efforts are needed. Regarding our example from Fig.

2, fitness value $Fit(S)$ of the original reference process model S corresponds to $Fit(S) = AC(S) \times SF(S) = 0.858 \times 0.749 = 0.643$.

5 Constructing the Search Tree

5.1 The Search Tree

Let us revisit Fig. 3, which gives a general overview of our heuristic search approach. Starting with the current candidate model S_c, in each iteration we search for its direct "neighbors" (i.e., process models which have distance 1 to S_c) trying to find a better candidate model S'_c with higher fitness value. Generally for a given process model S_c, we can construct a neighbor model by applying *ONE* insert, delete, or move operation to S_c. All activities $a_j \in \bigcup N_i$, which appear at least in one variant, are candidate activities for change. Obviously, an insert operation adds a new activity $a_j \notin N_c$ to S_c, while the other two operations delete or move an activity a_j already present in S_c (i.e., $a_j \in N_c$).

Generally, numerous process models may result by changing one particular activity a_j on S_c. Note that the positions where we can insert ($a_j \notin N_c$) or move ($a_j \in N_c$) activity a_j can be numerous. Section 5.2 provides details on how to find all process models resulting from the change of one particular activity a_j on S_c. First of all, we assume that we have already identified these neighbor models, including the one with highest fitness value (denoted as the *best kid* S^j_{kid} of S_c when changing a_j). Fig. 6 illustrates our search tree (see [8] for more details). Our search algorithm starts with setting the original reference model S as the initial state, i.e., $S_c = S$ (cf. Fig. 6). We further define AS as *active activity set*, which contains all activities that might be subject to change. At the beginning, $AS = \{a_j | a_j \in \bigcup_{i=1}^{n} N_i\}$ contains all activities that appear in at least one variant S_i. For each activity $a_j \in AS$, we determine the corresponding best kid S^j_{kid} of S_c. If S^j_{kid} has higher fitness value than S_c, we mark it; otherwise, we remove a_j from AS (cf. Fig. 6). Afterwards, we choose the model with highest

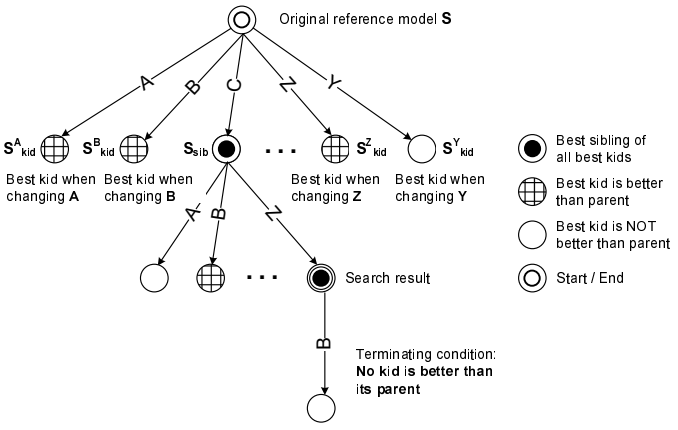

Fig. 6. Constructing the search tree

fitness value S^{j*}_{kid} among all best kids S^{j}_{kid}, and denote this model as *best sibling* S_{sib}. We then set S_{sib} as the first intermediate search result and replace S_c by S_{sib} for further search. We also remove a_{j*} from AS since it has been already considered.

The described search method goes on iteratively, until termination condition is met, i.e., we either cannot find a better model or the allowed search distance is reached. The final search result S_{sib} corresponds to our discovered reference model S' (the node marked by a bull's eye and circle in Fig. 6).

5.2 Options for Changing One Particular Activity

Section 5.1 has shown how to construct a search tree by comparing the best kids S^{j}_{kid}. This section discusses how to find such best kid S^{j}_{kid}, i.e., how to find all "neighbors" of candidate model S_c by performing *one* change operation on a particular activity a_j. Consequently, S^{j}_{kid} is the one with highest fitness value among all considered models. Regarding a particular activity a_j, we consider three types of basic change operations: *delete, move* and *insert* activity. The neighbor model resulting through deletion of activity $a_j \in N_c$ can be easily determined by removing a_j from the process model and the corresponding order matrix [7]; furthermore, movement of an activity can be simulated by its deletion and subsequent re-insertion at the desired position. Thus, the basic challenge in finding neighbors of a candidate model S_c is to apply one activity *insertion* such that *block structuring* and *soundness* of the resulting model can be guaranteed. Obviously, the positions where we can (correctly) insert a_j into S_c are our subjects of interest. Fig. 7 provides an example. Given process model S_c, we would like to find all process models that may result when inserting activity `X` into S_c. We apply the following two steps to "simulate" the insertion of an activity.

Step 1 (Block-enumeration): First, we enumerate all possible blocks the candidate model S_c contains. A block can be an atomic activity, a self-contained part of the process model, or the process model itself. Let S_c be a process model with activity set $N_c = \{a_1, \ldots, a_n\}$ and let further A_c be the order matrix of S_c. Two activities a_i and a_j can form a block if and only if $[\forall a_k \in N_c \setminus \{a_i, a_j\} : A_{ik} = A_{jk}]$ holds (i.e., iff they have exactly same order relations to the remaining activities). Consider our example from Fig. 7a. Here `C` and `D` can form a block {`C, D`} since they have same order relations to remaining activities `G, H, I` and `J`. In our context, we consider each block as set rather than as process model, since its structure is evident in S_c. As extension, two blocks B_j and B_k can merge into a bigger one iff $[(a_\alpha, a_\beta, a_\gamma) \in B_j \times B_k \times (N \setminus B_j \bigcup B_k) : A_{\alpha\gamma} = A_{\beta\gamma}]$ holds; i.e., two blocks can merge into a bigger block iff all activities $a_\alpha \in B_j$, $a_\beta \in B_k$ show same order relations to the remaining activities outside the two blocks. For example, block {`C, D`} and block {`G`} show same order relations in respect to remaining activities `H, I` and `J`; therefore they can form a bigger block {`C, D, J`}. Fig. 7a shows all blocks contained in S_c (see [8] for a detailed algorithm).

Step 2 (Cluster Inserted Activity with One Block): Based on the enumerated blocks, we describe where we can (correctly) insert a particular activity a_j in S_c. Assume that we want to insert `X` in S_c (cf. Fig. 7b). To ensure

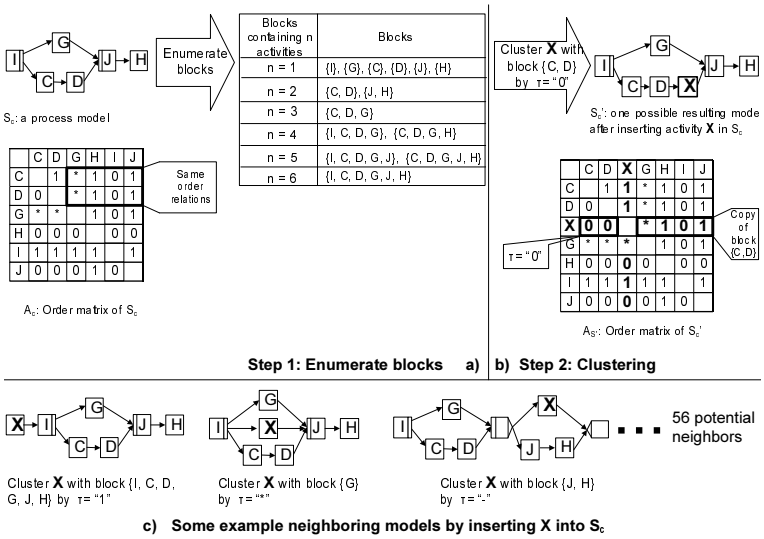

Fig. 7. Finding the neighboring models by inserting `X` into process model S

the block structure of the resulting model, we "cluster" `X` with an enumerated block, i.e., we replace one of the previously determined blocks B by a bigger block B' that contains B as well as `X`. In the context of this clustering, we set order relations between `X` and all activities in block B as $\tau \in \{0, 1, *, -\}$ (cf Def. 3). One example is given in Fig. 7b. Here inserted activity `X` is clustered with block {`C, D`} by order relation $\tau =$ "0", i.e., we set `X` as successor of the block containing `C` and `D`. To realize this clustering, we have to set order relations between `X` on the one hand and activities `C` and `D` from the selected block on the other hand to "0". Furthermore, order relations between `X` and the remaining activities are same as for {`C, D`}. Afterwards these three activities form a new block {`C, D, X`} replacing the old one {`C, D`}. This way, we obtain a sound and block-structured process model S_c' by inserting `X` into S_c.

We can guarantee that the resulting process model is sound and block-structured. Every time we cluster an activity with a block, we actually add this activity to the position where it can form a bigger block together with the selected one, i.e., we replace a self-contained block of a process model by a bigger one. Obviously, S_c' is not the only neighboring model of S_c. For each block B enumerated in Step 1, we can cluster it with `X` by any one of the four order relations $\tau \in \{0, 1, *, -\}$. Regarding our example from Fig. 7, S contains 14 blocks. Consequently, the number of models that may result when adding `X` to S_c equals $14 \times 4 = 56$; i.e., we can obtain 56 potential models by inserting `X` into S_c. Fig. 7c shows some neighboring models of S_c.

5.3 Search Result for our Running Example

Regarding our example from Fig. 2, we now present the search result and all intermediate models we obtain when applying our algorithm (see Fig. 8).

The first operation $\Delta_1 = move(S, \texttt{J,B}, endFlow)$ changes original reference model S into intermediate result model R_1 which is the one with highest fitness value among all neighboring models of S. Based on R_1, we discover R_2 by change $\Delta_2 = insert(R_1, \texttt{X, E, B})$, and finally we obtain R_3 by performing change $\Delta_3 = move(R_2, \texttt{I,D,H})$ on R_2. Since we cannot find a "better" process model by changing R_3 anymore, we obtain R_3 as final result. Note that if we only allow to change original reference model by maximal d change operations, the final search result would be: R_d *if* $d \leq 3$ *or* R_3 *if* $d \geq 4$.

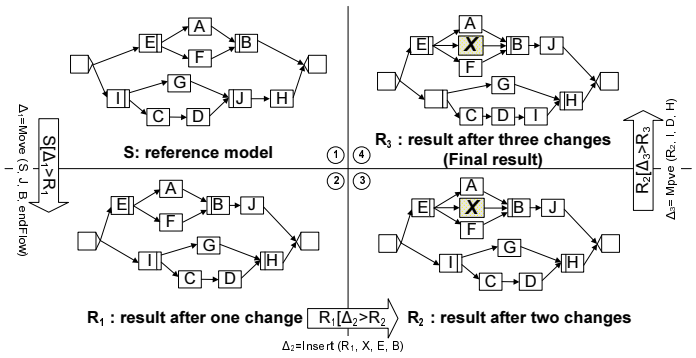

Fig. 8. Search result after every change

We further compare original reference model S and all (intermediate) search results in Fig. 9. As our heuristic search algorithm is based on finding process models with higher fitness values, we observe improvements of the fitness values for each search step. Since such fitness value is only a "reasonable guessing", we also compute average weighted distance between the discovered model and the variants, which is a precise measurement in our context. From Fig. 9, average weighted distance also drops monotonically from 4 (when considering S) to 2.35 (when considering R_3).

Additionally, we evaluate *delta-fitness* and *delta-distance*, which indicate relative change of fitness and average weighted distance for every iteration of the algorithm. For example, Δ_1 changes S into R_1. Consequently, it improves fitness value (delta-fitness) by 0.0171 and reduces average weighted distance (delta-distance) by 0.8. Similarly, Δ_2 reduces average weighted distance by 0.6 and Δ_3 by 0.25. The monotonic decrease of delta-distance indicates that important changes (reducing average weighted distance between reference model and variants most) are indeed discovered at beginning of the search.

Another important feature of our heuristic search is its ability to automatically decide on which activities shall be included in the reference model. A predefined threshold or filtering of the less relevant activities in the activity set is not needed; e.g., `X` is automatically inserted, but `Y` and `Z` are not added. The three change operations (insert, move, delete) are automatically balanced based on their influence on the fitness value.

5.4 Proof-of-Concept Prototype

The described approach has been implemented and tested using Java. We use our ADEPT2 Process Template Editor [12] as tool for creating process variants. For each process model, the editor can generate an XML representation with all relevant information (like nodes, edges, blocks). We store created variants in a variants repository which can be accessed by our mining procedure. The mining algorithm has been developed as stand-alone service which can read the original reference model and all process variants, and generate the result models according to the XML schema of the process editor. All (intermediate) search results are stored and can be visualized using the editor.

6 Simulation

Of course, using one example to measure the performance of our heuristic mining algorithm is far from being enough. Since computing average weighted distance is at $\mathcal{NP} - hard$ level, fitness function is only an approximation of it. Therefore, the first question is *to what degree delta-fitness is correlated with delta-distance?* In addition, we are interested in knowing to what degree important change operations are performed at the beginning. If biggest distance reduction is obtained with the first changes, setting search limitations or filtering out the change operations performed at the end, does not constitute a problem. Therefore, the second research question is: *To what degree are important change operations positioned at the beginning of our heuristic search?*

We try to answer these questions using *simulation*; i.e., by generating thousands of data samples, we can provide a statistical answer for these questions. In our simulation, we identify several parameters (e.g., size of the model, similarity of the variants) for which we investigate whether they influence the performance of our heuristic mining algorithm (see [8] for details). By adjusting these parameters, we generate 72 groups of datasets (7272 models in total) covering different scenarios. Each group contains a randomly generated *reference process model* and a collection of *100 different process variants*. We generate each variant by configuring the reference model according to a particular scenario.

We perform our heuristic mining to discover new reference models. We do not set constraints on search steps, i.e., the algorithm only terminates if no better model can be discovered. *All (intermediate) process models* are documented (see Fig. 8 as example). We compute the *fitness* and *average weighted distance* of each intermediate process models as obtained from our heuristic mining. We

	S	R_1	R_2	R_3
Fitness	0.643	0.814	0.854	0.872
Average weighted distance	4	3.2	2.6	2.35
Delta-fitness		0.171	0.04	0.017
Delta-distance		0.8	0.6	0.25

Fig. 9. Evaluation of the search results

	Execution time information			Correlation analysis		
	# of activity per variant	Average search steps	Average execution time (s)	# of data	Correlation	Significant?
Small-sized	$10 \sim 13$	1.83	0.148	33	0.762	Yes
Medium-sized	$20 \sim 26$	3.52	4.568	74	0.589	Yes
Large-sized	$50 \sim 65$	8.43	805.539	177	0.623	Yes

Fig. 10. Execution time and correlation analysis of groups with different sizes

additionally compute *delta-fitness* and *delta-distance* in order to examine the influence of every change operation (see Fig. 9 for an example).

Improvement on average weighted distances. In 60 (out of 72) groups we are able to discover a new reference model. The average weighted distance of the discovered model is *0.765* lower than the one of the original reference model; i.e., we obtain a reduction of *17.92%* on average.

Execution time. The number of activities contained in the variants can significantly influence execution time of our algorithm. Search space becomes larger for bigger models since the number of candidate activities for change and the number of blocks contained in the reference model become higher. The average run time for models of different size is summarized in Fig. 10.

Correlation of delta-fitness and delta-distance. One important issue we want to investigate is how delta-fitness is correlated to delta-distance. Every change operation leads to a particular change of the process model, and consequently creates a delta-fitness x_i and delta-distance y_i. In total, we have performed *284* changes in our simulation when discovering reference models. We use Pearson correlation to measure correlation between delta-fitness and delta-distance [13]. Let X be delta-fitness and Y be delta-distance. We obtain n data samples (x_i, y_i), $i = 1, \ldots, n$. Let $\bar{x}$ and $\bar{y}$ be the mean of X and Y, and let s_x and s_y be the standard deviation of X and Y. The Pearson correlation r_{xy} then equals $r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y}$ [13]. Results are summarized in Fig. 10. All correlation coefficients are *significant* and *high* (> 0.5). The high positive correlation between delta-fitness and delta-distance indicates that when finding a model with higher fitness value, we have very high chance to also reduce average weighted distance. We additionally compare these three correlations. Results indicate that they do not show significant difference from each other, i.e., they are statistically same (see [8]). This implies that our algorithm provides search results of similar goodness *independent* of the number of activities contained in the process variants.

Importance of top changes. Finally, we measure to what degree our algorithm applies more important changes at the beginning. In this context, we measure to what degree the top $n\%$ changes have reduced the average weighted distance. For example, consider search results from Fig. 9. We have performed in total 3 change operations and reduced the average weighted distance by 1.65 from 4 (based on S) to 2.35 (based on R_3). Among the three change operations, the first one reduces average weighted distance by 0.8. When compared to the overall distance reduction of 1.65, the top 33.33% changes accomplished 0.8/1.65

= 48.48% of our overall distance reduction. This number indicates how important the changes at beginning are. We therefore evaluate the distance reduction by analyzing the top 33.3% and 50.0% change operations. On average, the top 33.3% change operations have achieved *63.80%* distance reduction while the top 50.0% have achieved *78.93%*. Through this analysis, it becomes clear that the changes at beginning are *a lot more important* than the ones performed at last.

7 Related Work

Though heuristic search algorithms are widely used in areas like data mining [14] or artificial intelligence [10], only few approaches use heuristics for process variant management. In process mining, a variety of techniques have been suggested including heuristic or genetic approaches [19, 2, 16]. As illustrated in [6], traditional process mining is different from process variant mining due to its different goals and inputs. There are few techniques which allow to learn from process variants by mining recorded change primitives (e.g., to add or delete control edges). For example, [1] measures process model similarity based on change primitives and suggests mining techniques using this measure. Similar techniques for mining change primitives exist in the field of association rule mining and maximal sub-graph mining [14] as known from graph theory; here common edges between different nodes are discovered to construct a common sub-graph from a set of graphs. However, these approaches are unable to deal with silent activities and also do not differentiate between AND- and XOR-branchings. To mine high level change operations, [3] presents an approach using process mining algorithms to discover the execution sequences of changes. This approach simply considers each change as individual operation so the result is more like a visualization of changes rather than mining them. None of the discussed approaches is sufficient in supporting the evolution of reference process model towards an easy and cost-effective model by learning from process variants in a controlled way.

8 Summary and Outlook

The main contribution of this paper is to provide a heuristic search algorithm supporting the discovery of a reference process model by learning from a collection of (block-structured) process variants. Adopting the discovered model as new reference process model will make process configuration easier. Our heuristic algorithm can also take the original reference model into account such that the user can control how much the discovered model is different from the original one. This way, we cannot only avoid spaghetti-like process models but also control how many changes we want to perform. We have evaluated our algorithm by performing a comprehensive simulation. Based on its results, the fitness function of our heuristic algorithm is highly correlated with average weighted distance. This indicates good performance of our algorithm since the approximation value we use to guide our algorithm is nicely correlated to the real one. In addition, simulation results also indicate that the more important changes are performed at the beginning - the first 1/3 changes result in about 2/3 of overall distance

reduction. Though results look promising, more work needs to be done. As our algorithm takes relatively long time when encountering large process models, it would be useful to further optimize it to make search faster.

References

1. J. Bae, L. Liu, J. Caverlee, and W.B. Rouse. Process mining, discovery, and integration using distance measures. In *ICWS '06*, pages 479–488, 2006.
2. A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, NL, 2006.
3. C.W. Günther, S. Rinderle-Ma, M. Reichert, W.M.P. van der Aalst, and J. Recker. Using process mining to learn from process changes in evolutionary systems. *Int'l Journal of Business Process Integration and Management*, 3(1):61–78, 2008.
4. A. Hallerbach, T. Bauer, and M. Reichert. Managing process variants in the process lifecycle. In *Proc. 10th Int'l Conf. on Enterprise Information Systems (ICEIS'08)*, pages 154–161, 2008.
5. C. Li, M. Reichert, and A. Wombacher. Discovering reference process models by mining process variants. In *ICWS'08*, pages 45–53. IEEE Computer Society, 2008.
6. C. Li, M. Reichert, and A. Wombacher. Mining process variants: Goals and issues. In *IEEE SCC (2)*, pages 573–576. IEEE Computer Society, 2008.
7. C. Li, M. Reichert, and A. Wombacher. On measuring process model similarity based on high-level change operations. In *ER '08*, pages 248–262, 2008.
8. C. Li, M. Reichert, and A. Wombacher. A heuristic approach for discovering reference models by mining process model variants. Technical Report TR-CTIT-09-08, University of Twente, NL, 2009.
9. C. Li, M. Reichert, and A. Wombacher. What are the problem makers: Ranking activities according to their relevance for process changes. In *ICWS'09*, page to appear. IEEE Computer Society, 2009.
10. G. F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Pearson Education, 2005.
11. M. Reichert and P. Dadam. ADEPTflex -supporting dynamic changes of workflows without losing control. *J. of Intelligent Information Sys.*, 10(2):93–129, 1998.
12. M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with ADEPT2. In *ICDE '05*, pages 1113–1114. IEEE Computer Society, 2005.
13. D.J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, 2004.
14. P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.
15. W.M.P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203, 2002.
16. W.M.P van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE TKDE*, 16(9):1128–1142, 2004.
17. B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, 66(3):438–466, 2008.
18. B. Weber, M. Reichert, W. Wild, and S. Rinderle-Ma. Providing integrated life cycle support in process-aware information systems. *Int'l Journal of Cooperative Information Systems (IJCIS)*, 19(1), 2009.
19. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integr. Com.-Aided Eng.*, 10(2):151–162, 2003.